ENSC 650:
Data Analysis

Final Project:
Optical Character Recognition of Handwritten Persian Alphabet Letters

by:
Negar Hassanpour
230105914
Introduction

Handwriting recognition (or HWR) is the ability of a computer to receive and interpret handwritten input from sources such as paper documents, photographs, touch-screens and other devices. Even with introduction of new technologies, handwritten texts are ubiquitous in all human transactions and therefore machine recognition of handwriting has practical significance. For example, in postal addresses on envelopes, in amounts in bank checks, in handwritten fields in forms, etc (Plamondon, 2000).

Handwriting data is converted to digital form by scanning the writing on paper by optical scanning (Optical Character Recognition, or simply OCR). Alternatively, the movements of the pen tip may be sensed on an electronic surface such as a digitizer combined with a liquid crystal display. The two approaches are distinguished as off-line and on-line handwriting, respectively (Plamondon, 2000).

Handwriting recognition is difficult since different people have different handwriting styles. Although, narrowing the problem domain often helps increase the accuracy of handwriting recognition systems. A form field for a ZIP code for example, would contain only the characters alphabet letters and 0-9 digits. This fact would reduce the number of possible identifications.
It is necessary to perform some preprocessing operations of the text image prior to text recognition. Thresholding, or binarization is the very first preprocessing operation which is a task of converting a gray-scale image into a binary black-white image. Different methods such as histogram equalization may be used to get the best thresholding. An example for binarization is illustrated in Figure 1. It is clear that a thresholded image is much less noisy and if it is applied, the recognition accuracy would definitely increase.

Figure 1. Text Image Binarization

After converting the text image into black and white, it is necessary to normalize the size of the image in order to become comparable with the reference images in data base.

In the next step, the text image is divided into a number of zones which is called zoning. To do so, an n by m grid is superimposed on the character image

---

and the mean and variance of pixel in each zone is calculated to form a feature vector of that character image for later recognition (Due Trier, 1996). On left side of Error! Reference source not found., a 4x4 zoned picture is illustrated. On the right side, the average gray scale level of each zone is demonstrated in gray scale level. As stated earlier, this mean value along with the variance as diversity measure are serving to create the feature vector of each character image.

![Image zoning and feature extraction](image)

**Figure 2. Image zoning and feature extraction**

After feature extraction, a machine needs to be trained with a set training data containing observations whose category membership is known, and build an statistical model so that it can identify to which categories a new observation belongs. This is a statistical approach in text recognition in which the extracted features of each image are in ordered and fixed-length list of
numerical values. Statistical approach relies on firmly established elements of statistical decision theory (Fukunaga, 1990).

It is also good to mention that there is another approach on which is called structural text recognition. Features in this approach consist of unordered, variable-length list of simple "shapes". Although the structural approach is much closer to human recognition strategy, it is usually hard to implement in a fast, trainable, and robust way over a large variety of shapes (Heutte, 1998).

**Data set**

The data set used in this project contains thousands of images of 8 letters in Persian Alphabet. Figure 3 shows these 8 letters as well as their equivalence to English.

A total collection of 1800 hand-written samples (images) per each character is available. Each scanned image of the letters is threshold and cropped so that the margin of the new image meets the boundary of the letter. Then it is resized to the appropriate image size and zoned into 5x5 blocks. Mean and variance of each block is considered a feature of each character image. Therefore, we have 5x5x2 = 50 features for each character image.
After extracting the feature from all images inside the data base and constructing a data set of feature vectors, the data set is divided into two training and testing categories. 1400 samples per character are selected for training (constructing) the text recognition model and what remains is used for testing and validation of each proposed model.
Feature Conditioning

The first feature conditioning process that is applied to the data is whitening transformation which transforms the data set into a new decorrelated (covariance = 0) data set with zero mean and unit variance (variance = 1). The method is called "whitening" because it transforms the input matrix to the form of white noise. If $\Phi$ is the Eigen vector of a sample matrix $\Sigma$ with zeroed mean, and $\Lambda$ is its diagonal matrix of Eigen values, then the whitening transformation is defined as $A_w = \Phi \Lambda^{-\frac{1}{2}}$ (Duda, 2000). Figure 4 illustrates this transformation.

Whitening is essential for two reasons: first, it can help us to detect any linear relationship between features in data. Linear relationship is not
desirable since the feature(s) which is(are) linearly dependent on other features not only bring no added value to the recognition process, but also by making the feature matrix a singular matrix which basically incapacitates us to perform any further linear algebraic operation on our data. Secondly, it removes the bias (mean) and equalizes the distinguishability power (variance) of all features.

**Principle Component Analysis (PCA)**

The next step for feature conditioning is trying to represent the data in a lower-dimensional space instead of a huge 50-dimentional space. Although this may cause reduction of degrees of freedom, however in return, it reduces the computational complexity, also known as curse of dimensionality, in statistical techniques to pattern recognition problems. We can reduce the dimensionality from d-dimensions to for example one dimension if we merely project the d-dimensional data onto a line. Of course, by moving the line around, we might be able to find an orientation for which the projected samples are well separated (Duda, 2000).

Our data set is consisting of 8 classes, so we probably are interested to define a measure and tells us how much separable our data is with regarding
this set of features. The first thing to look at is to check how much the data within a class are close to each other. i.e. within-class scatter. With this measure, we calculate a summation of distances between each data inside a class to the mean of that class ($S_j$) and then sum this value for all classes. This brings us to $S_w$ which is proportional to the sample covariance matrix for the d-dimensional data. It must be symmetric and positive semi-definite (Here come the necessity of whitening and de-singularizing the data!). The second thing to be aware of is that how far each class is from other classes. This measure is calculated over comparing means of each class. Figure 5 shows how to calculate these two matrices.

$$S_w: \text{Within Scatter matrix} = \sum_{j=1}^{l} S_j, \quad S_j = \sum_{x^q \in c_j} (x^q - \mu_j)(x^q - \mu_j)^T$$

$$S_B: \text{Between Scatter matrix} = \sum_{j=1}^{l} N_j * (\mu_j - \mu)(\mu_j - \mu)^T$$

$$J = \text{Trace} (S_w^{-1} S_B) \quad \text{(Separability Measure)}$$

Figure 5. Definition of a separability measure

We define a new measure of J, which is proportional to $S_B$ and inversely proportional to $S_w$, since we would like the $S_B$ to be large and $S_w$ to be small. In order to maximize J, equation must be satisfied: $S_Bw = \lambda S_ww \rightarrow (S_w^{-1} S_B)w = \lambda w \rightarrow$ which is the well-known Eigen value, Eigen vector equation (Duda, 2000).
Whew! OK, now that we know our data set separability measure, we may want to reduce the dimension of our space. What we do is to sort the Eigen values (or singular values in case of a non-square matrix) in a descending order and hold as many dimensions as we want to preserve the separability. For example, the first 7 Eigen values out of all 50 Eigen values of \((S_w^{-1} S_B)\) matrix of our dataset are reported in Table 1. The rest of Eigen values were too close to zero. The sum of all Eigen values of \((S_w^{-1} S_B)\), also called as separability index, is equal to 21.6976.

<table>
<thead>
<tr>
<th>(\lambda_1)</th>
<th>(\lambda_2)</th>
<th>(\lambda_3)</th>
<th>(\lambda_4)</th>
<th>(\lambda_5)</th>
<th>(\lambda_6)</th>
<th>(\lambda_7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.6330</td>
<td>4.38465</td>
<td>1.96628</td>
<td>0.96575</td>
<td>0.48099</td>
<td>0.23903</td>
<td>0.02783</td>
</tr>
</tbody>
</table>

If we look close to the table above, we can observe that even if we only hold 4 dimensions out of 50 dimensions, we still have 96.5% separable data set in a 4 dimensional space which is far simpler than a 50 dimensional one!!

\[
100 \times \frac{\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4}{\sum_{i=1}^{50} \lambda_i} = 100 \times \frac{13.6331 + 4.3847 + 1.9663 + 0.9658}{21.6976} \approx 96.5\%
\]

Finally, here is where PCA takes place. This method is an approach towards finding the features from the data which best describes the variation, which respectively brings us to derive information out of data. Here we only want 4 dimensional data, therefore we multiply the 50 dimensional data by
the respective Eigen vectors of the above-mentioned Eigen values and ... done!

\(^{\wedge}_{\wedge}\)/

Though, I used all 7 non-zero values as for a the new dimension of my data and this way, the separability was preserved by more than 99.9%.

**Classification**

In this section, 3 classifiers are introduced and explained how they work. In this project Linear Regression, Bayes Optimal Classifier, and Neural Network are used for building a statistical machine learning model to recognize those 8 Persian alphabet hand-written letters.

In order that we can compare the results of these different classifiers together, we need to identify some evaluation criteria: Confusion Matrix and Correct Classification Rate (CCR).

Confusion Matrix is a table that allows visualization of the performance of a Supervised Learning algorithm. To make the concept clearer, I put an example here: Imagine we have trained a classifier to distinguish between cats, dogs and rabbits. After training the classifier, in order to test whether it has gained enough discrimination power or not, we for example show 8 unseen photos of cat to the classifier. Please note that we know the picture
belongs to which animal all the time, i.e. supervised learning; but the classifier has not seen that picture in training. The classifier correctly classifies 5 photos of cat as cat, but classifies 3 photos of cat as dog, and so on. Figure 6 shows the completed confusion matrix of this example.

![Confusion Matrix](image)

**Figure 6. an example for Confusion Matrix**

Correct Classification Rate (CCR) is a value in percentage which is derived from the confusion matrix. Clearly as stated above, the diagonal values in the confusion matrix are the correct predictions of the classifiers and summation of all values in the confusion matrix is the total number of evaluation questions. CCR is defined as:

\[
CCR = \frac{\text{no. of correct predictions}}{\text{no. of all samples}} = \frac{(5)+(3)+(11)}{(5+3+0)+(2+3+1)+(0+2+11)}
\]
The model with the higher CCR in evaluation (testing) is considered to be having the best accuracy.

**Linear Regression**

We already know that Linear Regression is a statistical method for modeling the relationship between a dependent variable denoted as $y$, and one or more explanatory variables denoted as $X$. It is widely used in the disciplines where there exist possible linear relationships between variables. Data in this method is modeled using linear predictor functions, and model parameters which are estimated from the data.

Linear regression can be used as a classification method as (Williams, 1998) stated. I used the features as explanatory variables or predictors ($X$) and the real class of data (a quantized value between 1 to 8 for each letter) as dependent variable or predictant ($y$). The red lines in Figure 7 illustrates the predictions of the Linear Regression classifier for evaluation (testing) data, for both 50 ($1^{st}$ graph) and 7 ($2^{nd}$ graph) number of features. The actual class is graphed in blue line so that the reader can get an idea of how linear regression has worked for this data.
As it is clear from the graphs, a linear method is not able to distinguish between different classes of such data. Table 2 and Table 3 show the Confusion Matrix of two implementations of Linear Regression Classifier, one with 50 features and the other with only 7 features.

Table 2. Confusion Matrix of Linear Regression Classifier using 50 features

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>183</th>
<th>162</th>
<th>40</th>
<th>2</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Class</td>
<td>103</td>
<td>206</td>
<td>67</td>
<td>12</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>23</td>
<td>181</td>
<td>157</td>
<td>28</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>4</td>
<td>60</td>
<td>206</td>
<td>119</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>9</td>
<td>80</td>
<td>240</td>
<td>68</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>21</td>
<td>137</td>
<td>188</td>
<td>49</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>48</td>
<td>197</td>
<td>133</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>27</td>
<td>86</td>
<td>164</td>
<td>93</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We would like to know whether the obtained regression model holds true for such and can be used to predict the class of each data point. In other words, we want to see whether the predictors (features) are linearly related to predictant (the right class) or not. To do so, we perform the statistical significance test and the results are extremely unexpected, since it shows that the regression is statistically significant!

\[
F_{calculated} = \frac{SSR/k}{SSE/(N - k - 1)} = \frac{46777/50}{12023/(11200 - 50 - 1)} = 867.532
\]

\[
F_{calculated} = 867.532 \quad \text{and} \quad F_{table, \alpha=0.05} = 1.394 \quad \rightarrow F_{calculated} > F_{table}
\]

\[
\rightarrow \text{Regression IS statistically significant!}
\]

**MATLAB:**
The hypothesis is that predictors X (features) are linearly unrelated to predictant y (correct classes). Hypothesis Rejected! Regression IS statistically significant with alpha=0.05
Bayes Optimal Classifier

The Bayes Optimal Classifier is an optimal classification technique which employs the probability theory for decision making. On average, no other ensemble can outperform it, so it is the ideal ensemble (Mitchell, 1997). The objective in Bayes Optimal Classifier is to find the most probable class regarding training data set.

In case of our recognition problem, during training, the model builds 8 (number of classes) Gaussian distributions in a 50 (number of features) or 7 dimensional space. The parameters of Gaussian distribution are derived from the training data samples. These Gaussian parameters are the mean and variance of each feature over all training data samples.

As for evaluation of this method, when a new unseen data sample is fed to the model, the probabilities of it being a member of each of those 8 classes is calculated based on each class’s Gaussian distribution, then the class which retrieves the maximum probability is identified as the predicted class.

Figure 8 on the left shows a 2-class & 2-dimensional Gaussian distribution and on the right shows those two distributions from above. It also shows the decision border of these two distributions which demonstrates where is the highest probability in each coordinate of the space.
The results of the Bayes optimal classifier are so much better than the linear regression method since it can handle a nonlinear mapping of data. It can be inferred that the reason is because Bayes uses a non-linear method to predict the classes. Also, it tries to minimize the probability of error (i.e. misclassification) and that is why it works so efficiently.

### Table 4. Confusion Matrix of Bayes Optimal Classifier using 50 features

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>372</th>
<th>26</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Class</td>
<td>80</td>
<td>318</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>293</td>
<td>88</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>357</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>246</td>
<td>85</td>
<td>51</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>345</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 3</td>
<td>346</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 2</td>
<td>370</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5. Confusion Matrix of Bayes Optimal Classifier using 7 features

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>346</th>
<th>53</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>326</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>341</td>
<td>47</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>38</td>
<td>353</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>256</td>
<td>53</td>
<td>78</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>65</td>
<td>322</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>40</td>
<td>11</td>
<td>331</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>372</td>
<td></td>
</tr>
</tbody>
</table>

The CCRs, and also advantages and disadvantages of this method over other proposed methods will be discussed in the “comparison” section.

**Neural Network**

Neural Network is basically a combination of simple processing elements which exhibit complex global behaviour. It was first introduced by Rosenblatt’s perceptron which played an important role in the history of machine learning. Many of his ideas were encapsulated in “Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms”, published in 1962. Rosenblatt’s work was criticized by Marvin Minsky, whose objections were published in the book “Perceptrons”, co-authored with Seymour Papert. This book was widely misinterpreted at the time as showing that neural networks were fatally flawed and could only learn solutions for linearly
separable problems. In fact, it only proved such limitations in the case of single-layer networks.

Unfortunately, however, this book contributed to the substantial decline in research funding for neural computing, a situation that was not reversed until the mid-1980s. Today, there are many hundreds, if not thousands, of applications of neural networks in widespread use, with examples in areas such as handwriting recognition and information retrieval being used routinely by millions of people (Bishop, 2006).

Typically each neural network is defined by 3 types of parameters:

1. Interconnection between different layers of neurons (feed-forward, feed-back, etc)
2. Learning process for updating weights (gradient descent, etc)
3. Activation function converting neuron’s weighted input to output (different sigmoid curves such as tanh, etc)

The basic architecture consists of three types of neuron layers: input, hidden, and output. A feed-forward version of this architecture is illustrated in Figure 9. In feed-forward networks, the signal flow is from input to output units, strictly in a feed-forward direction. The data processing can extend over multiple layers of units, but no feedback connections are present.
Recurrent networks contain feedback connections. Contrary to feed-forward networks, the dynamical properties of the network are important. In some cases, the activation values of the units undergo a relaxation process such that the network will evolve to a stable state in which these activations do not change anymore.

In other applications, the changes of the activation values of the output neurons are significant, such that the dynamical behaviour constitutes the output of the network\(^2\).

Almost all neural networks employ various gradient decadent methods to update their weights. Please refer to (Theodoridis, 2003; Bishop, 2006) for a more comprehensive explanation of how this procedure is done.

As for our problem, I applied a single hidden layer neural network with 50 input neurons (or 7, in case that PCA is applied), 50 neurons for the hidden layer and, 8 neurons for the output. I set the maximum number of epochs to train the network as 25 and set the performance goal as 0.01. This parameter is a measure error and we want it to be small. It is the summation of squared error between the real output and the output given by the network in the training session.

When the neural network is being trained, if for example a sample data from class 1 is fed to the neural network, the correct answer that is shown to the network is such that the first output neuron gets 1 and all other 7 get 0. And the same process applies for other classes: for example for training the network to recognize class 2, the second output neuron is set to 1 and all others set to 0, and so on.

Then in the evaluation session, when a new unseen data is fed to the trained neural network, the 8 output neurons return 8 values. The maximum neuron that returns the maximum value is identified as the predicted class of our neural network.
The confusion matrices for both 50 input layers (i.e. features), and 7 input layers are presented in Table 6 and Table 7.

As it is clear from the confusion matrices, the accuracy of neural network is the best compared to Bayes optimal classifier and linear regression as here we have more number of correct classified samples. A full comparison of these three methods will be presented in the next section of this report.

Table 6. Confusion Matrix of Neural Network Classifier using 50 features

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>Actual Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>358</td>
<td>41 1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>35 362</td>
<td>0 1 1 1 0 0 0 0</td>
</tr>
<tr>
<td>4 2 342</td>
<td>46 1 2 1 2 0 0 0</td>
</tr>
<tr>
<td>0 0 25 370</td>
<td>0 3 0 2 0 0 0 0</td>
</tr>
<tr>
<td>0 1 1 1 39 353</td>
<td>364 4 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 2 2 1 15 381</td>
<td>0 0 0 3 1 5 10 0</td>
</tr>
</tbody>
</table>

Table 7. Confusion Matrix of Neural Network Classifier using 7 features

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>Actual Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>345</td>
<td>52 1 0 1 0 1 0 0</td>
</tr>
<tr>
<td>55 341</td>
<td>0 0 1 2 0 1 0 0</td>
</tr>
<tr>
<td>4 0 336</td>
<td>51 1 5 1 2 0 0 0</td>
</tr>
<tr>
<td>0 1 38 354</td>
<td>1 3 0 3 0 0 0 0</td>
</tr>
<tr>
<td>0 2 0 0 263 326</td>
<td>65 11 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 3 1 3 59</td>
<td>330 7 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 4 1 0 46 370</td>
<td>0 0 0 3 9 4 12 0</td>
</tr>
</tbody>
</table>
Comparison between classifiers

From the CCRs obtained from these tests, we can see that the best result belongs to Neural Network, which it very versatile in learning highly non-linear data sets. After Neural Networks and in second place, the Bayes optimal classifier comes which also shows acceptable CCRs in evaluation. But the Linear Regression seems not to be efficient for this kind of data. Table 8 shows the CCRs for both 50 features, and 7 features, as well as accuracy results for training and testing sessions. We already know that the testing CCR shows the generalization power of the implemented method. However, it is good to know the accuracy of training CCR as well, so that we get a valid understanding of whether the implemented method is successful (and gives us the right answer) for the data that we have trained it with.

Table 8. Training and Testing Correct Classification Rates for three implemented methods

<table>
<thead>
<tr>
<th></th>
<th>Linear Regression</th>
<th>Bayes Optimal</th>
<th>Neural Network</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 features</td>
<td>40.71%</td>
<td>85.19%</td>
<td>92.51%</td>
</tr>
<tr>
<td>7 features</td>
<td>40.71%</td>
<td>83.61%</td>
<td>84.81%</td>
</tr>
<tr>
<td><strong>Testing</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 features</td>
<td>40.20%</td>
<td>82.72%</td>
<td>89.22%</td>
</tr>
<tr>
<td>7 features</td>
<td>40.40%</td>
<td>82.72%</td>
<td>83.28%</td>
</tr>
</tbody>
</table>
It is very important to note that although the training CCR is always larger than the testing CCR, it shouldn’t be very much larger. If this is the case, overfitting is happened in the sense that the system works pretty well for the already seen training data, but has not gained sufficient generalization power to predict the correct class for unseen testing data.

In order to prevent overfitting, usually the number of training samples is reduced; or in iterative methods such as neural networks, the number of epochs may be reduced and/or the performance goal is increased. It is true that if we do not give the neural network much time (number of epoches) so that the predicted time-series in the output cannot get so close to the actual time-series, we can reduce the overfitting. Also, by increasing the performance goal, we can obtain a less overfitted model.

Another issue to look at when we want to implement a classifier is the computational complexity and the cost that different methods have. Based on the values reported in Table 9 which demonstrate the simulation time (in seconds) needed for each of the three implemented classifiers to give their final decision, the neural network is the slowest one, i.e., has the most computational complexity. However, it also has the best accuracy among these three proposed recognition methods ... therefore, we always face a trade-off between the accuracy and the computational complexity.
Table 9. Simulation time needed for each of the three implemented classifiers

<table>
<thead>
<tr>
<th>Simulation Time (sec)</th>
<th>Linear Regression</th>
<th>Bayes Optimal</th>
<th>Neural Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 features</td>
<td>0.038</td>
<td>10.614</td>
<td>395.119</td>
</tr>
<tr>
<td>7 features</td>
<td>0.005</td>
<td>6.373</td>
<td>63.619</td>
</tr>
</tbody>
</table>

It is also good to mention that dimension reduction methods such as PCA can tremendously reduce the computational complexity. So, it is very essential that we first apply such methods to reduce the “curse of dimensionality” in our data set, and then go for finding the best possible accuracy within a certain amount of computational complexity.

Table 10. Pros and Cons for each classification method

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| **Linear Regression** | • Simple and Easy to Implement  
                     • Low Computational Complexity                           | • Cannot handle non-linear patterns                                 |
| **Bayes Optimal**   | • Requires a small amount of training data to estimate parameters       | • Strong feature independence assumptions                           |
| **Neural Network**  | • Can learn any non-linear pattern  
                     • Can be extremely robust  
                     • Has a super flexible structure                                | • Requires a large diversity of training data for real-world operation  
                     • High Computational Complexity                                 |
As you can see in Table 10 above, there is a much more of the trade-off issues to be met when we want to find the best solution for a classification problem.

Table 10 shows the advantages and disadvantages of each of these methods and gives better understanding of which kind of classifier to use in order to get the sufficiently good result when we are constrained to a certain amount of computational complexity.

**Conclusions**

The problem was to recognize characters in an image of a hand-written text. After pre-processing and feature extraction from each character image, a feature conditioning was performed on the data so that they reserve the maximum information. Classifiers were trained by both raw data and PCAed data and results were compared. Linear classifiers such as Linear Regression may not be helpful in all situations, i.e. dealing with non-linear data. Therefore, the need for non-linear classifier comes forward. Bayes classifier is a good statistical method for classification of this pattern recognition problem. However, because we have large amount of data, it`s more convenient to use Neural Network which always come up with better Accuracy.
References


MATLAB Codes:
MainCode.m

clc
clear all
close all
load Feature_Vector

n_feature=size(X_trn,1);
n_class=size(X_trn,3);
n_trndata=size(X_trn,2);
n_tstdata=size(X_tst,2);

% Data Whitening ----------------------------------------------------------
X_all=[X_trn(:,:,1);X_trn(:,:,2);X_trn(:,:,3);X_trn(:,:,4); ...
      X_trn(:,:,5);X_trn(:,:,6);X_trn(:,:,7);X_trn(:,:,8); ...
      X_tst(:,:,1);X_tst(:,:,2);X_tst(:,:,3);X_tst(:,:,4); ...
      X_tst(:,:,5);X_tst(:,:,6);X_tst(:,:,7);X_tst(:,:,8)];

n_X_all=size(X_all,1);
sigma=cov(X_all);
[U,S,V]=svd(sigma);
feature_dep=size(find(diag(S)<10^-5),1);
if feature_dep
   display(strcat(num2str(size(find(eig(sigma)<10^-5),1)), ' of the features is (are) linearly dependent on other features.'))
   X_all=X_all(:,1:n_feature-feature_dep);
end

n_feature=n_feature-feature_dep;

sigma=cov(X_all);
S=svd(sigma);
A = diag(S(1:n_feature-feature_dep).^(-0.5))*U(:,1:n_feature-feature_dep); % Principle Component Analysis --------------------------------------------
b = mean(X_all);
for i=1:n_X_all
   X_all_w(i,:)=A*(X_all(i,:)-b)';
end

X_trnW=zeros(n_feature,n_trndata,n_class);
X_tstW=zeros(n_feature,n_tstdata,n_class);
for i=1:n_class
   X_trnW(:,:,i)=X_all_w((i-1)*n_trndata+1:i*n_trndata,:);
end
for i=1:n_class
   X_tstW(:,:,i)=X_all_w(n_class*n_trndata+(i-1)*n_tstdata+1:n_class*n_trndata+i*n_tstdata,:);
end
Sw=zeros(n_feature);
Sb=zeros(n_feature);
for i=1:n_class
    muj_trn(i,:)=mean(X_trnW(:,:,i)');
muj_tst(i,:)=mean(X_tstW(:,:,i)');
    Sj(:,:,i)=cov(X_trnW(:,:,i)',1)*n_trndata;
end
for i=1:n_class
    Sw=Sw+Sj(:,:,i);
end
mu_trn=mean(muj_trn);
u_tst=mean(muj_tst);
for i=1:n_class
    Sb=Sb+n_trndata*(muj_trn(i,:)'-mu_trn')*(muj_trn(i,:)-mu_trn);
end
SM=inv(Sw)*Sb;
[V,D] = eig(SM);
[B,IX]=sort(real(diag(D)),'descend');
S=0;
counter=0;
while S<sum(real(diag(D)))*0.999
    counter=counter+1;
    S=S+B(counter);
end
V_R=V(:,IX(1:counter));
n_feature=counter;
for i=1:n_class
    X_trn_PCA(:,:,i)=((X_trnW(:,:,i)'-repmat(mu_trn,n_trndata,1))*V_R)';
    X_tst_PCA(:,:,i)=((X_tstW(:,:,i)'-repmat(mu_tst,n_tstdata,1))*V_R)';
end

% Linear Regression --------------------------------------------------------
fprintf('
->%s
', 'Linear Regression with 50 features');
figure('Name', 'Linear Regression with 50 features');
tic
[Confusion_Matrix_trn_lr, CCR_trn_lr, Confusion_Matrix_tst_lr, CCR_tst_lr] = LinearRegression(X_trn,X_tst);
elapsedTime_lr = toc;

% Linear Regression --------------------------------------------------------
fprintf('
->%s
', 'Linear Regression with 7 features');
figure('Name', 'Linear Regression with 7 features');
tic
[Confusion_Matrix_trn_lr_PCA, CCR_trn_lr_PCA, Confusion_Matrix_tst_lr_PCA, CCR_tst_lr_PCA] = LinearRegression(X_trn_PCA,X_tst_PCA);
elapsedTime_lr_PCA = toc;

% Bayes Optimal Classifier -----------------------------------------------
fprintf('
->%s
', 'Bayes Optimal Classifier with 50 features');
tic
[Confusion_Matrix_trn_bc, CCR_trn_bc, Confusion_Matrix_tst_bc, CCR_tst_bc] = BayesClassifier(X_trn,X_tst);
elapsedTime_bc = toc;

% premature
fprintf('
->%s
', 'Bayes Optimal Classifier with 7 features');
tic
[Confusion_Matrix_trn_bc_PCA, CCR_trn_bc_PCA, Confusion_Matrix_tst_bc_PCA, CCR_tst_bc_PCA] = BayesClassifier(X_trn_PCA,X_tst_PCA);
elapsedTime_bc_PCA = toc;

% MLP Neural Network
fprintf('
->%s
', 'MLP Neural Network with 50 features');
n_hide = 50;
n_epochs = 25;
tic
[Confusion_Matrix_trn_nn, CCR_trn_nn, Confusion_Matrix_tst_nn, CCR_tst_nn] = NeuralNetwork(X_trn,X_tst,n_hide,n_epochs);
elapsedTime_nn = toc;

% premature
fprintf('
->%s
', 'Neural Network with 7 features');
n_hide = 50;
n_epochs = 25;
tic
[Confusion_Matrix_trn_nn_PCA, CCR_trn_nn_PCA, Confusion_Matrix_tst_nn_PCA, CCR_tst_nn_PCA] = NeuralNetwork(X_trn_PCA,X_tst_PCA,n_hide,n_epochs);
elapsedTime_nn_PCA = toc;

functions...

LinearRegression.m

function [Confusion_Matrix_trn, CCR_trn, Confusion_Matrix_tst, CCR_tst] = LinearRegression(X_trn,X_tst)
load tables

n_feature=size(X_trn,1);
n_class=size(X_trn,3);
n_trndata=size(X_trn,2);
n_tstdata=size(X_tst,2);

X_all_trn=[X_trn(:,:,1);X_trn(:,:,2);X_trn(:,:,3);X_trn(:,:,4);...
  X_trn(:,:,5);X_trn(:,:,6);X_trn(:,:,7);X_trn(:,:,8)];
X_all_tst=[X_tst(:,:,1);X_tst(:,:,2);X_tst(:,:,3);X_tst(:,:,4);...
  X_tst(:,:,5);X_tst(:,:,6);X_tst(:,:,7);X_tst(:,:,8)];

N_trn = n_trndata*n_class;
N_tst = n_tstdata*n_class;
X_trn = [ones(N_trn,1), X_all_trn];
X_tst = [ones(N_tst,1), X_all_tst];
y_trn = [];
for i = 1:n_class
    y_trn = [y_trn; i*ones(n_trndata,1)];
end
y_tst = [];
for i = 1:n_class
    y_tst = [y_tst; i*ones(n_tstdata,1)];
end
k = n_feature;
reg_co = inv(X_trn' * X_trn) * X_trn' * y_trn;
y_trn_class = round(X_trn * reg_co);
SSE = (y_trn - y_trn_class)' * (y_trn - y_trn_class);
SST = sum((y_trn - mean(y_trn)).^2);
SSR = SST - SSE;
R_trn = sqrt(SSR/SST);

F = (SSR/k)/(SSE/(N_trn-k-1));  
if (F >= f_table(row,column))
    display('Hypothesis Rejected! Regression IS statistically significant');
else
    display('Hypothesis Accepted! Regression is NOT statistically significant');
end

y_tst_class = round(X_tst * reg_co);
SSE = (y_tst - y_tst_class)' * (y_tst - y_tst_class);
SST = sum((y_tst - mean(y_tst)).^2);
SSR = SST - SSE;
R_tst = sqrt(SSR/SST);

for i=1:n_class
    for j=1:n_class
        Confusion_Matrix_trn(i,j) = length(find(y_trn_class((i-1)*n_trndata+1:i*n_trndata) == j));
        Confusion_Matrix_tst(i,j) = length(find(y_tst_class((i-1)*n_tstdata+1:i*n_tstdata) == j));
    end
end
CCR_trn=(sum(diag(Confusion_Matrix_trn)))/(sum(sum(Confusion_Matrix_trn))')*100;
CCR_tst=(sum(diag(Confusion_Matrix_tst)))/(sum(sum(Confusion_Matrix_tst))')*100;

PlotIt(y_trn,y_trn_class,y_tst,y_tst_class)
BayesClassifier.m

function [Confusion_Matrix_trn, CCR_trn, Confusion_Matrix_tst, CCR_tst] = BayesClassifier(X_train,X_test)

n_feature=size(X_train,1);
n_class=size(X_train,3);
n_trndata=size(X_train,2);
n_tstdata=size(X_test,2);

mu=mean(X_train,2);
for i=1:n_class
    sigma(:,:,i)=cov(X_train(:,:,i)');
end

for i=1:n_class
    for j=1:size(X_train,2)
        for k=1:n_class
            f(i,j,k)=mvnpdf(X_train(:,j,i)',mu(:,:,k)',sigma(:,:,k));
        end
        [~,INDICE(i,j)]=max(f(i,j,:));
    end
end

Confusion_Matrix_trn=zeros(n_class);
for i=1:n_class
    for j=1:n_class
        Confusion_Matrix_trn(i,j)=size(find(INDICE(i,:)==j),2);
    end
end
CCR_trn=(sum(diag(Confusion_Matrix_trn)))/(sum(sum(Confusion_Matrix_trn))')*100;

INDICE = [];
for i=1:n_class
    for j=1:size(X_test,2)
        for k=1:n_class
            f(i,j,k)=mvnpdf(X_test(:,j,i)',mu(:,:,k)',sigma(:,:,k));
        end
        [~,INDICE(i,j)]=max(f(i,j,:));
    end
end

Confusion_Matrix_tst=zeros(n_class);
for i=1:n_class
    for j=1:n_class
        Confusion_Matrix_tst(i,j)=size(find(INDICE(i,:)==j),2);
    end
end
CCR_tst=(sum(diag(Confusion_Matrix_tst)))/(sum(sum(Confusion_Matrix_tst))')*100;

NeuralNetwork.m

function [Confusion_Matrix_trn, CCR_trn, Confusion_Matrix_tst, CCR_tst] = NeuralNetwork(X_trn,X_tst,n_hide,n_epochs)
n_feature=size(X_trn,1);
n_class=size(X_trn,3);
n_trndata=size(X_trn,2);
n_tstdata=size(X_tst,2);

L = n_class;

X_all_trn=[X_trn(:,:,1)';X_trn(:,:,2)';X_trn(:,:,3)';X_trn(:,:,4)'; ...
    X_trn(:,:,5)';X_trn(:,:,6)';X_trn(:,:,7)';X_trn(:,:,8)'];
X_all_tst=[X_tst(:,:,1)';X_tst(:,:,2)';X_tst(:,:,3)';X_tst(:,:,4)'; ...
    X_tst(:,:,5)';X_tst(:,:,6)';X_tst(:,:,7)';X_tst(:,:,8)'];

y_all_trn = zeros(n_trndata*n_class,n_class);
y_all_tst = zeros(n_tstdata*n_class,n_class);
for i = 1:n_class
    y_all_trn((i-1)*n_trndata+1:i*n_trndata,i) = ones(n_trndata,1);
    y_all_tst((i-1)*n_tstdata+1:i*n_tstdata,i) = ones(n_tstdata,1);
end

net = newff(minmax(X_all_trn'),[n_hide,L],{'tansig' 'purelin'},'trainlm');
net.trainParam.epochs = n_epochs;
net.trainParam.goal = 1e-2;
[net,tr] = train(net,X_all_trn',y_all_trn');
y_trn = sim(net,X_all_trn');
y_tst = sim(net,X_all_tst');
[~, y_trn_class] = max(y_trn,[],1);
[~, y_tst_class] = max(y_tst,[],1);

for i=1:n_class
    for j=1:n_class
        Confusion_Matrix_trn(i,j) = length(find(y_trn_class((i-1)*n_trndata+1:i*n_trndata) == j));
        Confusion_Matrix_tst(i,j) = length(find(y_tst_class((i-1)*n_tstdata+1:i*n_tstdata) == j));
    end
end
CCR_trn=(sum(diag(Confusion_Matrix_trn)))/(sum(sum(Confusion_Matrix_trn))')*100;
CCR_tst=(sum(diag(Confusion_Matrix_tst)))/(sum(sum(Confusion_Matrix_tst))')*100;